Kauno technologijos universitetas

Informatikos fakultetas

Duomenų apdorojimas ir analizė

P176B101 Intelektikos pagrindai

Pirmas laboratorinis darbas

Autorius
Gustas Klevinskas

Akademinė grupė
IFF-8/7

Vadovai
Lekt. dr. Audrius Nečiūnas
Doc. dr. Agnė Paulauskaitė-Tarasevičienė

Kaunas, 2021

Turinys

Įvadas

Pirmojo laboratorinio darbo tikslas –pasirinkti duomenų rinkinį ir išmokti atlikti duomenų analizę, naudojant Python programavimo kalbą. Šiam tikslui įgyvendinti iškelti uždaviniai:

1. Rasti tinkamą duomenų rinkinį.
2. Sutvarkyti duomenis.
3. Atlikti duomenų rinkinio kokybės analizę.
4. Nubraižyti atributų histogramas.
5. Identifikuoti duomenų kokybės problemas.
6. Programiškai realizuoti kokybinių problemų sprendimą.
7. Nustatyti ryšius tarp atributų naudojant įvairius vizualizacijos būdus.
8. Apskaičiuoti kovariacijos ir koreliacijos reikšmes.

1. Duomenų atributų analizė

Nuoroda į duomenų rinkinį: https://www.kaggle.com/bobbyscience/league-of-legends-diamond-ranked-games-10-min.

Pasirinktas duomenų rinkinys – kompiuterinio žaidimo „League of Legends“ pirmųjų 10 minučių statistika. Rinkinį sudaro beveik 10 tūkst. duomenų. Jis sugeneruotas automatiškai, t. y. duomenys gauti tiesiai iš žaidimo, o ne atlikus žaidėjų apklausą, tad šio rinkinio kokybė yra gera – nėra trūkstamų reikšmių ir pan. Tačiau mokymosi tikslams programiškai įgyvendintas ir blogų reikšmių atpažinimas bei šalinimas.

1 lentelė. *Duomenų rinkinio atributai*

Atributas	Tipas	Prasmė	Pavyzdys
Won	Kategorinis	Nurodo laimėjusią komandą.	red
First blood	Kategorinis	Komanda, pirmoji nužudžiusi priešininką.	blue
Kills	Tolydinis	Komandos nužudymų skaičius.	9
Deaths	Tolydinis	Komandos mirčių skaičius.	11
Assists	Tolydinis	Komandos narių skaičius, prisidėjęs prie nužudymo.	19
Dragons	Kategorinis	Nurodo komandos nugalabytų drakonų skaičių.	0
Heralds	Kategorinis	Komandos nudėto didelio monstro nudėjimų kiekis.	1
Towers	Kategorinis	Priešininkų nugriautų bokštų skaičius.	3
Total gold	Tolydinis	Komandos narių pinigų suma.	14712
Total minions killed	Tolydinis	Nudėtų pakalikų skaičius.	186

1 lentelėje pavaizduoti pakoreguoti duomenų rinkinio atributai. Originaliame duomenų rinkinyje buvo išvestinių bei perteklinių atributų, kurie buvo automatiškai pašalinti laboratorinio darbo programa.

Paprastesniam vaizdavimui, lentelėje pateikti sumažintas atributų skaičius – nuo „Kills“ iki „Total minions killed“ atskirų komandų atributai sujungti į vieną. Atliekant skaičiavimus atskirai žiūrima, pavyzdžiui, į „Blue kills“ ir „Red kills“, taip nurodant skirtinų komandų rezultatus.

2. Tolydiniai atributai

2 lentelėje pateikti tolydiniai atributai. Kaip minėta anksčiau, duomenys yra labai tvarkingi ir nėra trūkstamų reikšmių. Tačiau mokymosi tikslams vis tiek buvo atsižvelgta į trūkstamas reikšmes. Ranka įrašius blogas reikšmes (žodžius) į „blue_deaths“ stulpelį, programa jas teisingai aptinka – tai matoma iš mažesnio eilučių kiekio palyginus su kitais stulpeliais.

2 lentelė. *Tolydinių atributų charakteristikos*

Atributo pavadinimas	Kiekis	Trūkstamos reikšmės, %	Kardina-lumas	Minimali reikšmė	Maksimali reikšmė	1-asis kvarti-lis	3-asis kvarti-lis	Vidurkis	Mediana	Standartinis nuokrypis
blue_kills	9879	0	21	0	22	4	8	6.184	8	3.011
blue_deaths	9875	0	21	0	22	4	8	6.137	4	2.934
blue_assists	9879	0	30	0	29	4	9	6.645	10	4.065
blue_total_gold	9879	0	4739	10730	23701	15415	17459	16503.46	17828	1535.447
blue_total_minions_killed	9879	0	148	90	283	202	232	216.7	229	21.858
red_kills	9879	0	21	0	22	4	8	6.138	6	2.934
red_deaths	9879	0	21	0	22	4	8	6.184	8	3.011
red_assists	9879	0	28	0	28	4	9	6.662	3	4.061
red_total_gold	9879	0	4732	11212	22732	15427	17419	16489.04	16786	1490.888
red_total_minions_killed	9879	0	153	107	289	203	233	217.349	231	21.912

3 lentelė. *Kategorinių atributų charakteristikos*

Atributo pavadinimas	Kiekis	Trūkstamos reikšmės, %	Kardinalumas	Moda	Modos dažnumas	Moda, %	2-oji moda	2-osios modos dažnumas	2-oji moda, %
won	9876	0	2	red	4946	50.1	blue	4930	49.9
first_blood	9879	0	2	blue	4987	50.5	red	4892	49.5
blue_dragons	9879	0	2	0	6303	63.8	1	3576	36.2
blue_heralds	9879	0	2	0	8022	81.2	1	1857	18.8
blue_towers_destroyed	9879	0	5	0	9415	95.3	1	429	4.3
red_dragons	9879	0	2	0	5798	58.7	1	4081	41.3
red_heralds	9879	0	2	0	8298	84	1	1581	16
red_towers_destroyed	9879	0	3	0	9483	96	1	367	3.7

3 lentelėje pateikta kategorinių atributų analizė. Dauguma šių atributų – tolydinės reikšmės, tačiau dėl labai mažo kardinalumo, jos analizuotos kaip kategorinės. Kaip ir tolydiniuose duomenyse, į „won“ stulpelį buvo ranka įrašytos kelios blogos reikšmės. Tai matoma iš mažesnio duomenų skaičiaus. Trūkstamų reikšmių procentas nepasikeitė, nes 3 sugadintos reikšmės beveik nedaro įtakos rinkiniui.

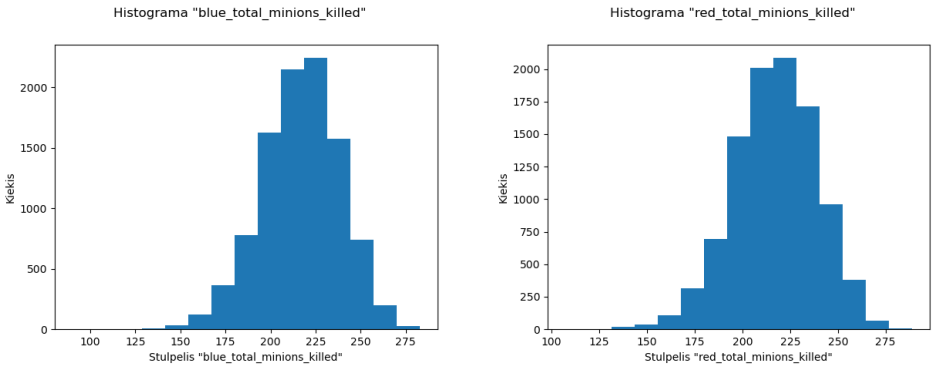
3. Duomenų grafinis atvaizdavimas ir analizė

3.1. Histogramos

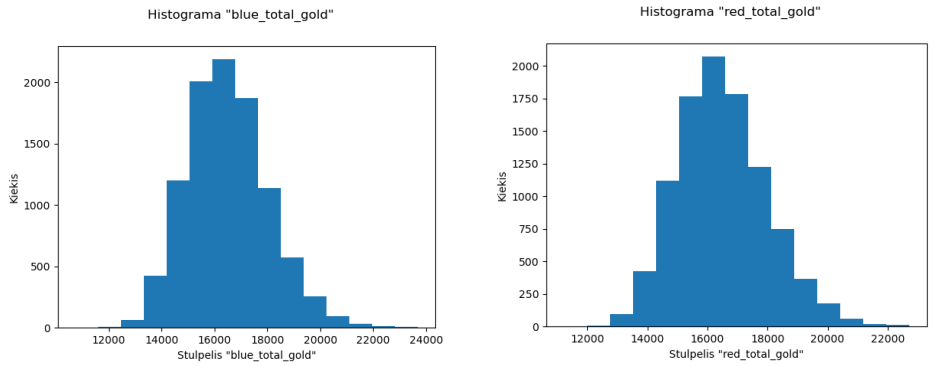
1 – 5 paveiksliukuose pateiktos tolydinių atributų histogramos. Šio duomenų rinkinio kategorinių atributų kardinalumas yra mažas – beveik visose histogramose būtų matomi tik du stulpeliai, tad palaikant tvarką ataskaitoje jie nepridėti. Trūkstamas histogramas galima sugeneruoti paleidus laboratorinio darbo programą.

Histogramų stulpelių kiekio apskaičiavimui naudota (1) formulė, kur w – stulpelio plotis, n – duomenų imties dydis.

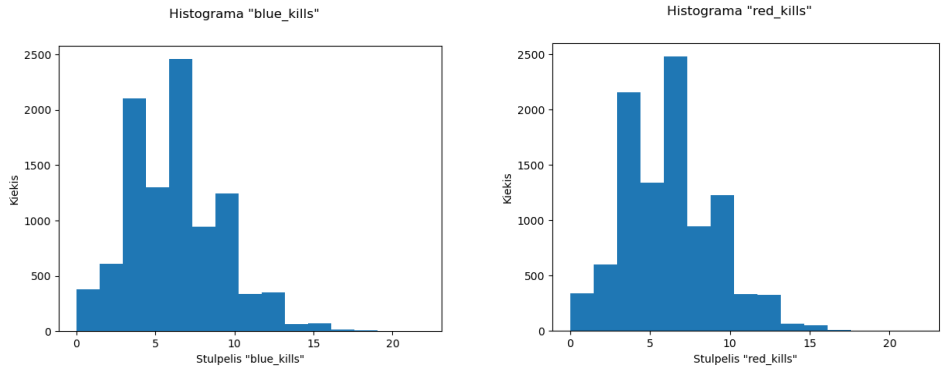
$$w = 1 + 3.22 \cdot \log_2(n) \tag{1}$$



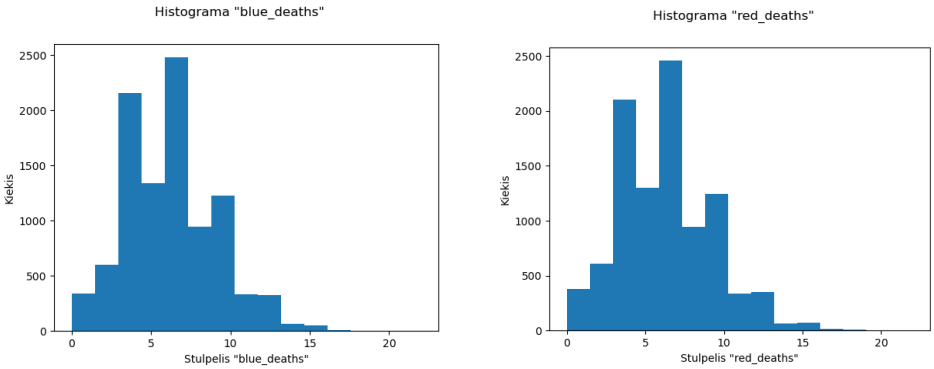
1 pav. *Nužudytų pakalikų histograma*



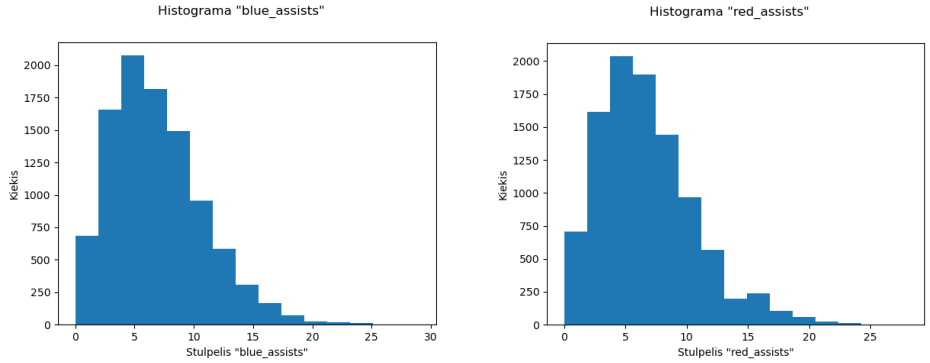
2 pav. *Komandos turimų pinigų histograma*



3 pav. *Nužudymų histograma*

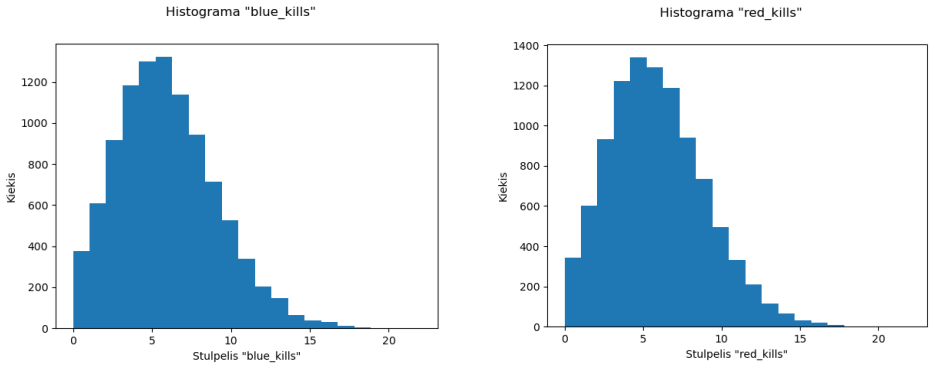


4 pav. *Mirčių histograma*

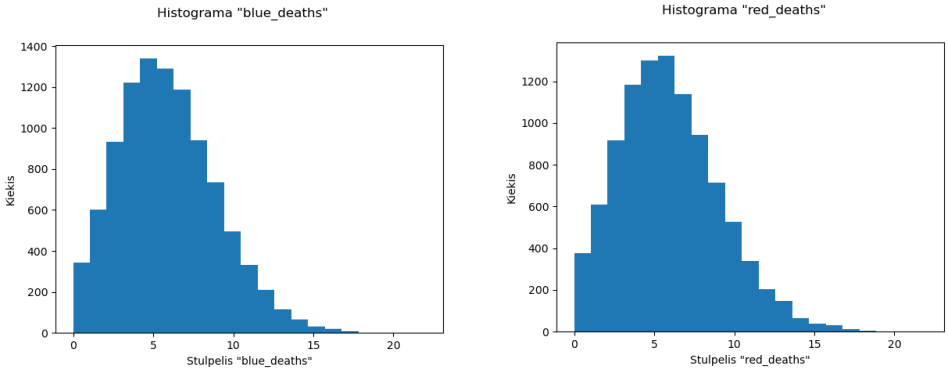


5 pav. *Pagelbėjimų histograma*

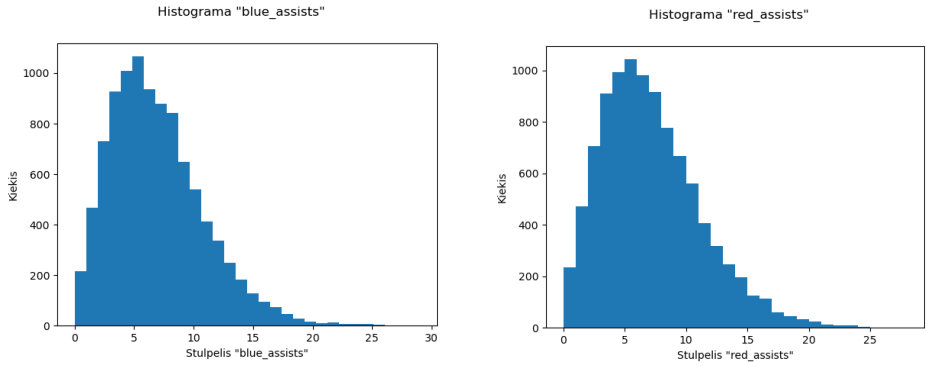
Dėl palyginus mažo kardinalumo, 3 – 5 pav. histogramos nėra korektiškos. Pavaizdavus negrupuojant duomenų gaunamos informatyvesnės diagramos, pateiktos 6 – 8 pav.



6 pav. *Pakoreguota nužudymų histograma*



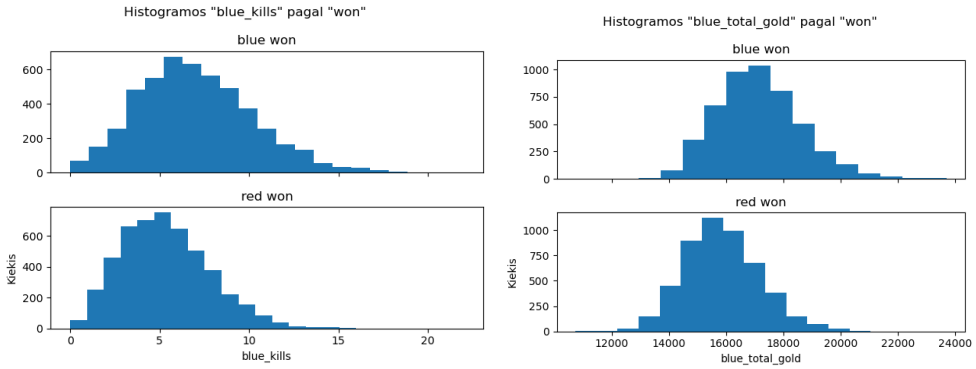
7 pav. *Pakoreguota mirčių histograma*



8 pav. *Pakoreguota pagelbėjimų histograma*

Visuose grafikuose matoma, jog dažniausiai pasitaiko vienas tam tikras intervalas, ir aplink jį vis tolstant mažėja atvejų. Iš to galima spręsti, jog histogramose duomenys pasiskirstę pagal normalųjį skirstinį.

8 pav. galima pamatyti, jog ties tam tikrais intervalais kiekis supanašėja ir nutolsta nuo normaliojo skirstinio. Tai matyti „blue assists“ ties 8 reikšme ir „red assists“ ties 15. Šis įvertis – nugalabijimui padėjusių žaidėjų skaičius. Kadangi tai komandinis žaidimas, tam tikri padėjimų skaičiai pasitaiko dažniau.

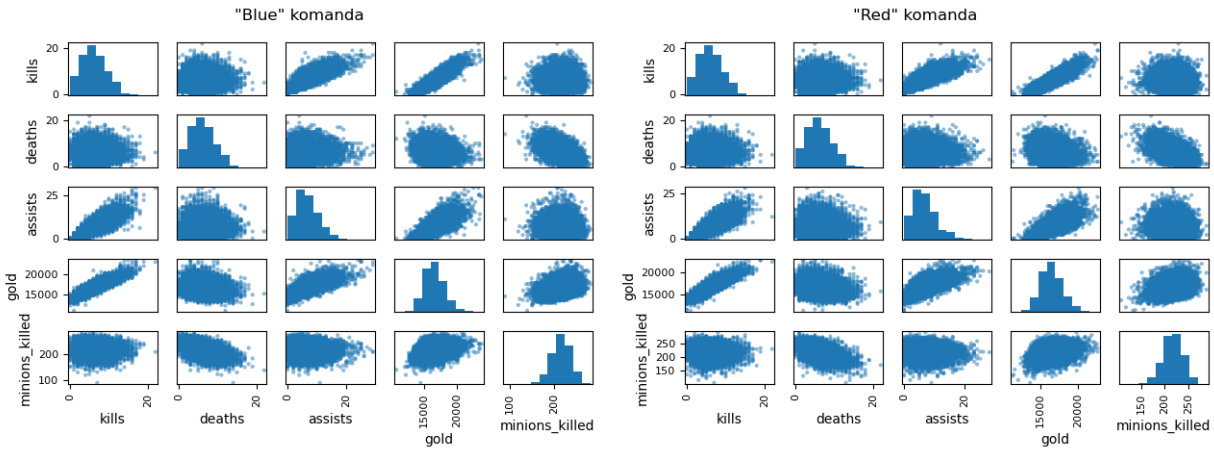


9 pav. *Kategorinio ir tolydinio atributų histogramos*

9 pav. pavaizduotos histogramos, rodančios ryšį tarp dviejų atributų: kairėje – mėlynos komandos nužudymų skaičiaus ir laimėjusios komandos, dešinėje – mėlynos komandos turimų pinigų bei laimėjusios komandos. Pirmojoje diagramoje matomas ryšys – komanda labiau tikėtina, jog laimės, turėdama kuo didesnį nužudymų skaičių. Toks pat ryšys matomas ir su turimais pinigais – kuo komanda turtingesnė, tuo ji gali išleisti daugiau žaidimo valiutos įvairiausiems daiktas, padedantiems laimėti.

3.2. Taškinės diagramos

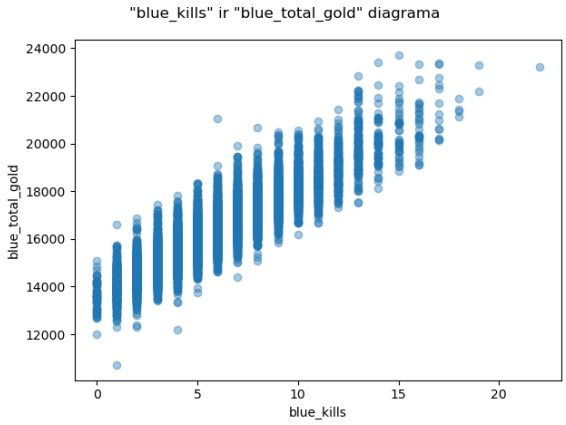
10 pav. pateiktos mėlynos ir raudonos komandų atributų „SPLOM“ diagramos. Analizavimui galima naudoti vieną iš jų, nes abiejų komandų atributų priklausomumas yra labai panašus – skiriasi tik kelios ribinės reikšmės. Iš to galima daryti išvadą, jog žaidimas yra gerai subalansuotas.



10 pav. *Mėlynos ir raudonos komandų „SPLOM“ diagramos*

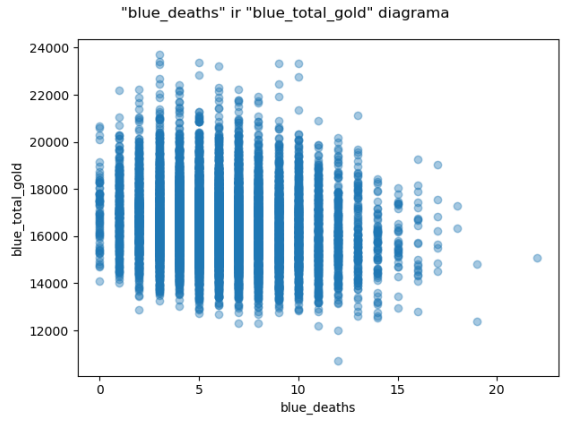
Diagramose galima įžvelgti du pagrindinius pasiskirstymo tipus: tiesiogiai priklausančius ir sudarančius vieną didelį klasterį.

11 ir 12 pav. pateiktos detalesnės diagramos tiesioginei priklausomybei pavaizduoti. Iš 11 pav. matyti, jog didėjant nudėjimų skaičiui didėja ir turimi pinigai. Pinigų diapazonas yra gana platus, nes pajamos gaunamos ne tik iš kitų žaidėjų žudymų.



11 pav. *Žudymo ir pinigų „scatter plot“ diagrama*

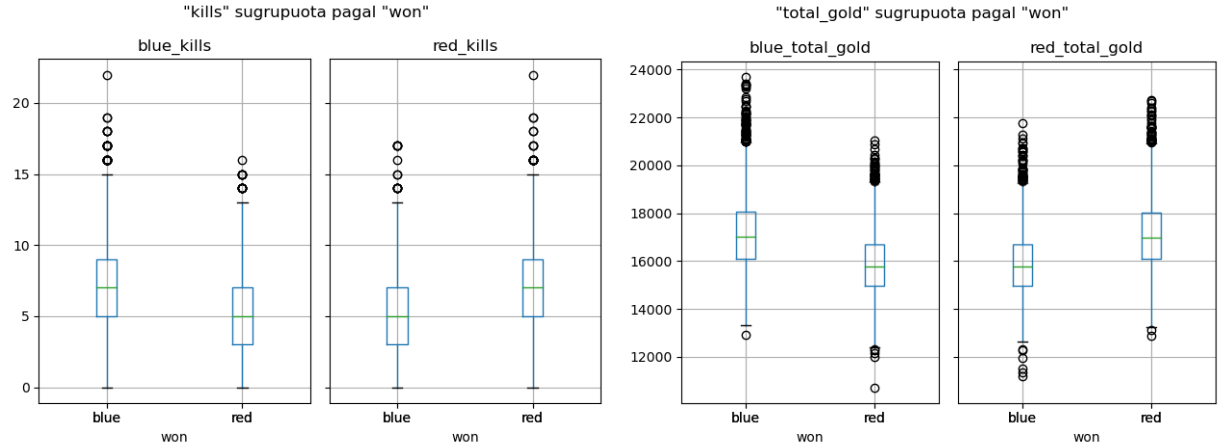
Silpną neigiamą koreliaciją galima pamatyti tarp mirčių ir aukso bei tarp mirčių ir nužudytų pakalikų. Iš šios priklausomybės, net nežinant apie žaidimą, galima įtarti, jog kol žaidėjas miręs, jis negali dobti pakalikų ir pamažėja gaunamos pajamos.



12 pav. *Mirčių ir pinigų „scatter plot“ diagrama*

3.3. „Box plot“ diagramos

13 pav. pateiktos „box plot“ tipo diagramos. Jos sugrupuotos pagal žaidimą laimėjusią komandą, nes tai yra pagrindinis šio duomenų rinkinio atributas. Esant poreikiui, programa gali pateikti stulpelio reikšmės sugrupuotas ir pagal kitas kategorines reikšmes.



13 pav. *„Box plot“ diagramos*

13 pav. matomas anksčiau minėtas faktas, jog laimi komandos, turinčios didesnį nužudymų skaičių. Toks pat faktas matomas ir dešiniajame pav.

Toks duomenų pavaizdavimas vizualiai parodo išsiskiriančias reikšmes. Pasirinkus jas apdoroti, labai stipriai pakeičiamas duomenų rinkinys. Tai aptariama detaliau 5 skyriuje.

4. Kovariacija ir koreliacija

4 bei 5 lentelėse atitinkamai pateiktos atributų koreliacijos ir kovariacijos reikšmės. Spalvomis sužymėti stulpeliai ir eilutės priklauso atitinkamai komandai.

4 lentelė. *Tolydinių atributų kovariacijos lentelė*

	Kills	**Deaths**	**Assists**	**Gold**	**Minions killed**	**Kills**	**Deaths**	**Assists**	**Gold**	**Minions killed**
Kills	9.07	0.03	9.96	4109.92	-2.03	0.03	9.07	-0.25	-723.57	-31.15
Deaths	0.03	8.61	-0.31	-732.8	-30.05	8.61	0.03	9.58	3875.48	-2.59
Assists	9.96	-0.31	16.52	4671.31	-5.52	-0.31	9.96	-0.12	-811.65	-30.06
Gold	4109.92	-732.8	4671.31	2357977	9556.34	-732.8	4109.92	-804.32	-719351	-15225.9
Minions killed	-2.03	-30.05	-5.52	9556.34	477.63	-30.05	-2.03	-29.94	-14594.9	0.27
Kills	0.03	8.61	-0.31	-732.8	-30.05	8.61	0.03	9.58	3875.48	-2.59
Deaths	9.07	0.03	9.96	4109.92	-2.03	0.03	9.07	-0.25	-723.57	-31.15
Assists	-0.25	9.58	-0.12	-804.32	-29.94	9.58	-0.25	16.49	4458.73	-6.95
Gold	-723.57	3875.48	-811.65	-719351	-14594.9	3875.48	-723.57	4458.73	2223454	9107.45
Minions killed	-31.15	-2.59	-30.06	-15225.9	0.27	-2.59	-31.15	-6.95	9107.45	480.17

5 lentelė. *Tolydinių atributų koreliacijos lentelė*

	Kills	Deaths	Assists	Gold	Minions killed	Kills	Deaths	Assists	Gold	Minions killed
Kills	1	0	0.81	0.89	-0.03	0	1	-0.02	-0.16	-0.47
Deaths	0	1	-0.03	-0.16	-0.47	1	0	0.8	0.89	-0.04
Assists	0.81	-0.03	1	0.75	-0.06	-0.03	0.81	-0.01	-0.13	-0.34
Gold	0.89	-0.16	0.75	1	0.28	-0.16	0.89	-0.13	-0.31	-0.45
Minions killed	-0.03	-0.47	-0.06	0.28	1	-0.47	-0.03	-0.34	-0.45	0
Kills	0	1	-0.03	-0.16	-0.47	1	0	0.8	0.89	-0.04
Deaths	1	0	0.81	0.89	-0.03	0	1	-0.02	-0.16	-0.47
Assists	-0.02	0.8	-0.01	-0.13	-0.34	0.8	-0.02	1	0.74	-0.08
Gold	-0.16	0.89	-0.13	-0.31	-0.45	0.89	-0.16	0.74	1	0.28
Minions killed	-0.47	-0.04	-0.34	-0.45	0	-0.04	-0.47	-0.08	0.28	1

14 pav. *Koreliacijos matrica*

14 pav. pateikta koreliacijos matrica padeda geriau suprasti 4 ir 5 lentelėse pateiktus duomenis atvaizduodama juos spalvomis. Iš jos galima pamatyti, jog analizuojami pertekliniai duomenys. Pavyzdžiui, „blue kills“ ir „red deaths“ koreliacijos įvertis yra 1 – mėlynai komandai atlikus nužudymą, raudonai komandai prisideda mirtis.

Ši diagrama parodo, jog yra stipri koreliacija tarp vienos komandos mirčių ir kitos komandos gaunamų pinigų. Stipriausi neigiami ryšiai – komandos mirčių ir pakalikų nužudymų skaičius. Kitaip tariant, didėjant mirčių skaičiui nužudoma mažiau pakalikų. Koreliacijos rodiklis patvirtina anksčiau analizuotas diagramas išreikšdamas pastebėjimus konkrečiais skaičiais.

5. Duomenų kokybė

5.1. Problemos

Šis duomenų rinkinys sudarytas automatiškai, pasinaudojant istoriniais žaidimų įrašais, o ne atliekant apklausas, kurioje žmonės ranka įvedinėja duomenis. Tad šis duomenų rinkinys neturi trūkstamų reikšmių ar dėl žmogaus klaidos sukurtų ekstremalių reikšmių. Mokymosi tikslui buvo ranka pakeistos kelios reikšmės. Stulpeliuose „won“ ir „first blood“ – pridėtos kitokios reikšmės, nei „blue“ ar „red“ komandų pavadinimai, į skaitines reikšmes parašyta teksto.

Originaliame duomenų rinkinyje yra daug išvadinių duomenų, tad jų tvarkymui parašyta Python programa, kuri modifikuoja duomenis tolimesniam laboratorinio darbo atlikimui ir naudojimo programoje.

Pagrindinė problema – iš pirmos pažiūros kaip tolydiniai atrodantys atributai, kurių kardinalumas labai mažas („dragons“, „heralds“ ir „towers destroyed“). Be to, iš 13 pav. „box plot“ diagramos matomas didelis kiekis ekstremalių reikšmių, tačiau jos ne dėl įvedimo klaidų.

5.2. Sprendimo planas

5.1 skyrelyje paminėtomis problemoms spręsti programiškai įgyvendintas ekstremalių reikšmių pašalinimas naudojant „IQR“ metodą.

$$IQR = Q_3 - Q_1 \tag{2}$$

$$upper = Q_3 + 1.5 \cdot IQR \tag{3}$$

$$lower = Q_1 - 1.5 \cdot IQR \tag{4}$$

2 – 4 formulėse pateiktas ribų apskaičiavimas, kur Q_1 – pirmasis kvartilis, Q_3 – trečiasis kvartilis. Reikšmės, esančios už šių ribų, laikomos ekstremaliomis. Šiame duomenų rinkinyje labai daug reikšmių patenka už šio intervalo, tad kažkaip modifikuojant jas stipriai pasikeistų statistika.

Pašalinus visas eilutes su ekstremaliomis reikšmėmis nebelieka eilučių. Tad šis apdorojimo būdas netinkamas. Įrašius vietoj tokios reikšmės imties modą vis tiek lieka didžiulis skaičius ribą peržengiančių reikšmių, šį skaičiavimą reiktų kartoti kelis kartus, o tai stipriai pakeistų duomenų rinkinį.

Ranka pakeistų reikšmių apdorojimui pasirinkta ištrinti nekorektiškas eilutes – duomenų rinkinyje yra daug eilučių, tad jų pašalinimas nedarys didelės įtakos statistikai. 5 formulėje pateikti skaitinių reikšmių apribojimai – jos negali būti neigiamos ir priklausyti sveikųjų skaičių aibei. Kategoriniai stulpeliai „won“ ir „first blood“ gali turėti tik tokius komandų pavadinimus: „blue“ ir „red“.

$$X \geq 0, \qquad X \in \mathbb{N} \tag{5}$$

6. Duomenų modifikavimas

6.1. Normalizacija

Skaitinėms reikšmėms normalizuoti pasitelkta „range“ normalizacija, kuri pateikta (6) formulėje.

$$Y_i = \frac{X_i - \min(X)}{\max(X) - \min(X)} \tag{6}$$

Nors šis būdas yra jautresnis ekstremalioms reikšmėms, šiame duomenų rinkinyje jie nedaro didelės blogos įtakos skaičiavimams ir duomenų vizualiam pateikimui. Normuojant „range“ metodu duomenys nebus papildomai iškraipomi.

6.2. Kategorinio tipo atributų vertimas į tolydinius

Beveik visi duomenų rinkinio atributai – skaitinės reikšmės, tad jų papildomai versti nereikia. Tik stulpelius „won“ bei „first blood“ reiktų pakeisti į vienos komandos statistiką, pavyzdžiui, „blue won“ ar „red first blood“. Šią binarinę būseną būtų galima išreikšti 0 arba 1, simbolizuojančias specifinės komandos pralaimėjimą ar pergalę.

7. Išvados

Iš atliktos analizės matyti, jog didelio skirtumo tarp komandos spalvos nėra – tai simbolizuoja, jog žaidimas „League of Legends“ yra gerai subalansuotas komandų atžvilgiu. Iš grafiškai pavaizduotų duomenų – yra stipri teigiama koreliacija tarp komandos nužudymų skaičiaus ir jos gautų pajamų. Silpna atvirkštinė koreliacija – tarp mirčių skaičiaus ir nužudytų pakalikų skaičiaus.

Paišant „box plot“ tipo diagramą parodoma daug ekstremalių reikšmių, tačiau „IQR“ ekstremumų šalinimo metodas šiam duomenų rinkiniui nėra pats tinkamiausias, dėl to, kad bus labai keičiama statistika nuo originalių reikšmių, arba nebeliks eilučių analizavimui, jei bus šalinamos eilutės su ekstremaliais įverčiais.

8. Programinis kodas

8.1. modify_data.py

```
import csv


INPUT_FILE = 'data.original.csv'
OUTPUT_FILE = 'data.csv'
DISCARD_HEADERS = [
    'gameId',
    'blueWins',
    'blueWardsPlaced',
    'blueWardsDestroyed',
    'blueFirstBlood',
    'blueEliteMonsters',
    'blueAvgLevel',
    'blueTotalExperience',
    'blueTotalJungleMinionsKilled',
    'blueGoldDiff',
    'blueExperienceDiff',
    'blueCSPerMin',
    'blueGoldPerMin',
    'redWardsPlaced',
    'redWardsDestroyed',
    'redFirstBlood',
    'redEliteMonsters',
    'redAvgLevel',
    'redTotalExperience',
    'redTotalJungleMinionsKilled',
    'redGoldDiff',
    'redExperienceDiff',
    'redCSPerMin',
    'redGoldPerMin'
]


if __name__ == '__main__':
    data = open(INPUT_FILE)
    modified_data = open(OUTPUT_FILE, mode='w')

    reader = csv.reader(data)
    writer = csv.writer(modified_data)

    header = next(reader)
    discard_ids = [idx for idx, i in enumerate(header) if i in DISCARD_HEADERS]

    new_header = [i for i in header if i not in DISCARD_HEADERS]
    new_header.insert(0, 'won')
    new_header.insert(1, 'firstBlood')

    writer.writerow(new_header)

    for row in reader:
        won = 'blue' if row[1] == '1' else 'red'
        firstBlood = 'blue' if row[4] == '1' else 'red'

        new_row = [i for idx, i in enumerate(row) if idx not in discard_ids]
        new_row.insert(0, won)
        new_row.insert(1, firstBlood)

        writer.writerow(new_row)
```

8.2. main.py

```
import csv
import re
from pathlib import Path

import pandas as pd
import seaborn as sn
from matplotlib import pyplot as plt
from pandas.plotting import scatter_matrix

from calc import *
from helper import *

INPUT_FILE = 'data.csv'
INPUT_LIMIT = -1  # -1 to disable
NUMERICAL_COLUMNS = [
    'blue_kills',
    'blue_deaths',
    'blue_assists',
    'blue_total_gold',
    'blue_total_minions_killed',
    'red_kills',
    'red_deaths',
    'red_assists',
    'red_total_gold',
    'red_total_minions_killed'
]
CATEGORICAL_COLUMNS = [
    'won',
    'first_blood',
    'blue_dragons',
    'blue_heralds',
    'blue_towers_destroyed',
    'red_dragons',
    'red_heralds',
    'red_towers_destroyed'
]


def read_data():
    data = open(INPUT_FILE, mode='r')
    reader = csv.DictReader(data)
    count = 0

    columns = {
        'won': [],
        'first_blood': [],
        'blue_kills': [],
        'blue_deaths': [],
        'blue_assists': [],
        'blue_dragons': [],
        'blue_heralds': [],
        'blue_towers_destroyed': [],
        'blue_total_gold': [],
        'blue_total_minions_killed': [],
        'red_kills': [],
        'red_deaths': [],
        'red_assists': [],
        'red_dragons': [],
        'red_heralds': [],
        'red_towers_destroyed': [],
        'red_total_gold': [],
        'red_total_minions_killed': [],
    }

    for row in reader:
        columns['won'].append(row['won'])
        columns['first_blood'].append(row['firstBlood'])
        columns['blue_kills'].append(row['blueKills'])
        columns['blue_deaths'].append(row['blueDeaths'])
        columns['blue_assists'].append(row['blueAssists'])
        columns['blue_dragons'].append(row['blueDragons'])
        columns['blue_heralds'].append(row['blueHeralds'])
        columns['blue_towers_destroyed'].append(row['blueTowersDestroyed'])
        columns['blue_total_gold'].append(row['blueTotalGold'])
```

```
        columns['blue_total_minions_killed'].append(row['blueTotalMinionsKilled'])
        columns['red_kills'].append(row['redKills'])
        columns['red_deaths'].append(row['redDeaths'])
        columns['red_assists'].append(row['redAssists'])
        columns['red_dragons'].append(row['redDragons'])
        columns['red_heralds'].append(row['redHeralds'])
        columns['red_towers_destroyed'].append(row['redTowersDestroyed'])
        columns['red_total_gold'].append(row['redTotalGold'])
        columns['red_total_minions_killed'].append(row['redTotalMinionsKilled'])

        count += 1
        if INPUT_LIMIT != -1 and count >= INPUT_LIMIT:
            break

    return columns


def create_numerical_table(columns):
    Path('out/').mkdir(parents=True, exist_ok=True)
    file = open('out/numerical.csv', mode='w')
    writer = csv.writer(file)

    writer.writerow([
        'Atributo pavadinimas',
        'Kiekis',
        'Trūkstamos reikšmės, %',
        'Kardinalumas',
        'Minimali reikšmė',
        'Maksimali reikšmė',
        '1-asis kvartilis',
        '3-asis kvartilis',
        'Vidurkis',
        'Mediana',
        'Standartinis nuokrypis'
    ])

    for key in NUMERICAL_COLUMNS:
        col = get_fixed_column(columns, key)

        writer.writerow([
            key,
            len(col),
            '{:.1f}'.format(get_missing_percent(columns, key)),
            cardinality(col),
            min(col),
            max(col),
            first_quartile(col),
            third_quartile(col),
            '{:.3f}'.format(avg(col)),
            median(col),
            '{:.3f}'.format(std_dev(col))
        ])


def create_categorical_table(columns):
    Path('out/').mkdir(parents=True, exist_ok=True)
    file = open('out/categorical.csv', mode='w')
    writer = csv.writer(file)

    writer.writerow([
        'Atributo pavadinimas',
        'Kiekis',
        'Trūkstamos reikšmės, %',
        'Kardinalumas',
        'Moda',
        'Modos dažnumas',
        'Moda, %',
        '2-oji moda',
        '2-osios modos dažnumas',
        '2-oji moda, %',
    ])

    for key in CATEGORICAL_COLUMNS:
        col = get_fixed_column(columns, key)
        m1, m1_freq, m1_percent = mode(col)
        m2, m2_freq, m2_percent = mode2(col)
```

```
        writer.writerow([
            key,
            len(col),
            '{:.1f}'.format(get_missing_percent(columns, key)),
            cardinality(col),
            m1,
            m1_freq,
            '{:.1f}'.format(m1_percent),
            m2,
            m2_freq,
            '{:.1f}'.format(m2_percent)
        ])


def create_table(columns, fun):
    Path('out/').mkdir(parents=True, exist_ok=True)
    file = open('out/{}.csv'.format(fun.__name__), mode='w')
    writer = csv.writer(file)

    writer.writerow(['', *NUMERICAL_COLUMNS])

    for key1 in NUMERICAL_COLUMNS:
        row = [key1]

        for key2 in NUMERICAL_COLUMNS:
            row.append('{:.2f}'.format(fun(columns, key1, key2)))

        writer.writerow(row)


def draw_histogram(columns, key):
    plt.figure('Histograma "{}"'.format(key))
    plt.suptitle('Histograma "{}"'.format(key))
    plt.xlabel('Stulpelis "{}"'.format(key))
    plt.ylabel('Kiekis')
    plt.hist(columns[key], bins='sturges')


def draw_scatter_plot(columns, key1, key2):
    plt.figure('"{}" ir "{}" diagrama'.format(key1, key2))
    plt.suptitle('"{}" ir "{}" diagrama'.format(key1, key2))
    plt.scatter(columns[key1], columns[key2], alpha=0.4)
    plt.xlabel(key1)
    plt.ylabel(key2)


def draw_splom(columns, team):
    result = {}

    for key in columns:
        if team in key and key in NUMERICAL_COLUMNS:
            new_key = re.sub('blue_|red_|total_', '', key)
            result[new_key] = columns[key]

    scatter_matrix(pd.DataFrame(result))
    plt.suptitle('"{}" komanda'.format(team.capitalize()))


def draw_hist_by_won(columns, key):
    fig, axs = plt.subplots(2, sharex='all')
    fig.suptitle('Histogramos "{}" pagal "won"'.format(key))

    won = []
    lost = []
    for idx, i in enumerate(columns['won']):
        if i == 'blue':
            won.append(columns[key][idx])
        else:
            lost.append(columns[key][idx])

    axs[0].hist(won, bins='sturges')
    axs[1].hist(lost, bins='sturges')

    plt.xlabel(key)
    plt.ylabel('Kiekis')
```

```
    axs[0].set_title('blue won')
    axs[1].set_title('red won')


def draw_box_plot(columns, key, key_by='won'):
    pd_columns = ['blue_' + key, 'red_' + key]
    df = pd.DataFrame(columns)
    df.boxplot(column=pd_columns, by=key_by)
    plt.suptitle('"{}" sugrupuota pagal "{}"'.format(key, key_by))


def draw_correlation_matrix(columns):
    plt.figure('Koreliacijos matrica')
    plt.suptitle('Koreliacijos matrica')

    df = pd.DataFrame(columns, columns=NUMERICAL_COLUMNS)
    matrix = df.corr()
    sn.heatmap(matrix)


def normalize_data(columns):
    norm = {}

    for key in columns:
        if key in NUMERICAL_COLUMNS:
            min_v = min(columns[key])
            width = max(columns[key]) - min_v
            norm[key] = [(i - min_v) / width for i in columns[key]]

            # mean = avg(columns[key])
            # dev = std_dev(columns[key])
            # norm[key] = [(i - mean) / dev for i in columns[key]]
        else:
            norm[key] = columns[key]

    return norm


def cat_to_num(columns):
    result = {}

    for key in columns:
        if key in ['won', 'first_blood']:
            result['blue_' + key] = map(lambda cell: 1 if cell == 'blue' else 0, columns[key])
        else:
            result[key] = columns[key]

    return result


def main():
    columns = read_data()

    # 1. Create column data analysis tables
    create_numerical_table(columns)
    create_categorical_table(columns)

    # 2. Data manipulation
    columns = get_fixed_columns(columns)
    # columns = remove_outliers(columns, NUMERICAL_COLUMNS)
    # columns = cat_to_num(columns)
    # columns = normalize_data(columns)

    # 3. Draw histograms
    # for key in columns:
    #     draw_histogram(columns, key)

    # 4. Draw scatter plot matrix
    draw_splom(columns, 'blue')
    draw_splom(columns, 'red')

    # 5. Draw scatter plots
    draw_scatter_plot(columns, 'blue_kills', 'blue_total_gold')
    draw_scatter_plot(columns, 'blue_deaths', 'blue_total_gold')
```

```
    # 6. Draw histograms by 'won'
    draw_hist_by_won(columns, 'blue_kills')
    draw_hist_by_won(columns, 'blue_total_gold')

    # 7. Draw box plots
    draw_box_plot(columns, 'kills')
    draw_box_plot(columns, 'total_gold')

    # 8. Calculate covariance and correlation
    create_table(columns, covariance)
    create_table(columns, correlation)
    draw_correlation_matrix(columns)

    plt.show()


if __name__ == '__main__':
    main()
```

8.3. helper.py

```
from copy import deepcopy

import calc


def __is_team_value_bad(value):
    return value != 'blue' and value != 'red'


def __is_numerical_value_bad(value):
    try:
        if int(value) < 0:
            return True
    except ValueError:
        return True

    return False


def is_value_bad(key, value):
    if key == 'won' or key == 'first_blood':
        return __is_team_value_bad(value)
    else:
        return __is_numerical_value_bad(value)


def get_frequencies(column):
    freq = {}

    for cell in column:
        freq[cell] = freq.get(cell, 0) + 1

    return freq


def get_fixed_columns(columns):
    result = deepcopy(columns)
    bad_rows = set()

    for key in result:
        for row_id, cell in enumerate(result[key]):
            if is_value_bad(key, cell):
                bad_rows.add(row_id)
            elif key != 'won' and key != 'first_blood':
                result[key][row_id] = int(result[key][row_id])

    for key in result:
        for row_id in reversed(range(len(result[key]))):
            if row_id in bad_rows:
                del result[key][row_id]

    return result


def get_fixed_column(columns, key):
    result = []

    for cell in columns[key]:
        if not is_value_bad(key, cell):
            if key != 'won' and key != 'first_blood':
                result.append(int(cell))
            else:
                result.append(cell)

    return result


def remove_outliers(columns, numerical):
    result = {}

    for key in columns:
        if key in numerical:
            median = calc.median(columns[key])
            q1 = calc.first_quartile(columns[key])
```

```
        q3 = calc.third_quartile(columns[key])
        iqr = q3 - q1

        result[key] = []

        for cell in columns[key]:
            if 1.5 * iqr + q3 >= cell >= 1.5 * iqr - q1:
                result[key].append(cell)
            else:
                result[key].append(median)
    else:
        result[key] = columns[key]

return result
```

8.4. calc.py

```
from math import sqrt

from helper import is_value_bad, get_frequencies


def get_missing_percent(columns, key):
    bad_count = 0

    for cell in columns[key]:
        if is_value_bad(key, cell):
            bad_count += 1

    return bad_count / len(columns[key]) * 100


def cardinality(column):
    freq = get_frequencies(column)
    return len(freq)


def first_quartile(column):
    column = sorted(column)
    n = len(column)
    return median(column[:n // 2])


def third_quartile(column):
    column = sorted(column)
    n = len(column)

    if n % 2 != 0:
        return median(column[n // 2 + 1:])
    else:
        return median(column[n // 2:])


def median(column):
    n = len(column)

    if n % 2 != 0:
        return column[n // 2]
    else:
        return (column[n // 2 - 1] + column[n // 2]) / 2


def avg(column):
    return sum(column) / len(column)


def std_dev(column):
    mean = avg(column)
    total = 0

    for i in column:
        total += (i - mean) ** 2

    return sqrt(total / (len(column) - 1))


def mode(column):
    freq = get_frequencies(column)
    max_key = max(freq, key=freq.get)

    return max_key, freq[max_key], freq[max_key] / len(column) * 100


def mode2(column):
    freq = get_frequencies(column)
    max_cell = max(freq, key=freq.get)
    second_cell = 0

    for cell in column:
        if cell != max_cell:
            second_cell = cell
```

```
            break

    for cell in freq:
        if freq[second_cell] < freq[cell] and cell != max_cell:
            second_cell = cell

    return second_cell, freq[second_cell], freq[second_cell] / len(column) * 100


def covariance(columns, key1, key2):
    if len(columns[key1]) != len(columns[key2]):
        raise ValueError('columns are not equal length')

    n = len(columns[key1])
    key1_avg = avg(columns[key1])
    key2_avg = avg(columns[key2])
    total = 0

    for i in range(n):
        total += (columns[key1][i] - key1_avg) * (columns[key2][i] - key2_avg)

    return total / (n - 1)


def correlation(columns, key1, key2):
    return covariance(columns, key1, key2) / (std_dev(columns[key1]) * std_dev(columns[key2]))
```